Introduction to C

CMSC 104, Fall 2012
John Y. Park

Introduction to C

Topics

- Brief History of Programming Languages & C
- The Anatomy of a C Program
- Compilation
- Using the gcc Compiler
- 104 C Programming Standards and Indentation Styles

History of Programming Languages & C

- Machine code (aka “binary”)
  - Somehow enter raw sequence of binary patterns
    1011010111001011
    1011010110101010
- Assembly “language”
  - Gave human-friendly syntax to machine code:

```
MOV   1200, R0
SUB   1202, R0
MOV   R0, 1200
```

History of Programming Languages & C

- Early high-level languages
  - COBOL
    - SUBTRACT B FROM A GIVING C
    - MULTIPLY C BY 2 GIVING D
  - FORTRAN

```
S1 = 3.0
S2 = 4.0
H = SQRT((S1 * S1) + (S2 * S2))
```


History of Programming Languages & C

- Another early high-level language
  - LISP

```
(lambda (a)
        (mapcar (func '+)
          (cons (car (car a)) (car (cadr a)))))
```


History of C

- Derived from… (wait for it…) "B"!
  - ("B" itself was derived from the BCPL language)
- Design goals were for C to be:
  - **Efficient**
  - **Close to the machine**
    - I.e., it could directly manipulate the CPU's memory to control hardware-level functions
  - S**tructured**
    - A true high-level language with sophisticated control flow, data structures
    - Has goto's—but probably will never use them!

History of C

- UNIX was recoded in C
  - PDP-11 was a machine with 64 **Kilobytes** of addressable memory
    - (my laptop has 60,000x the memory!)
- C is written in C!
  - Of course, first versions were written in Assembler
  - Ritchie had great inspiration for a Trojan horse

Does Programming Language Choice Matter?

- Short answer: "Yes, but…"
- C:

```
main( ) {
  printf("hello, world");
}
```

- COBOL:

```
MAIN SECTION
DISPLAY "hello, world"
STOP RUN.
```

- Fortran77:

```
PROGRAM HELLO
PRINT*, 'hello, world'
END
```

- Lisp:

```
(defun helloworld ()
  (print "hello, world") )
```

- English:
  - Hello, world.
- Spanish:
  - Hola mundo
- French:
  - Salut le Monde
- Greek:
  - Γεια σου κόσμε

Writing C Programs

- A programmer uses a **text editor (not the same as a word processor!)** to create or modify files containing C code.
- Code is also known as **source code**.
- A file containing source code is called a **source file**.

A Simple C Program

- One of the first C program examples

```
1. int main ( ) {
2.     printf (“hello, world”) ;
3. }
```


A Simple C Program… to a Computer

m	a	i	n	(	)	{	\n	\t	p	r	i	n	f	(	“
h	e	l	l	o	,		w	o	r	l	d	“	)	;	\n
}	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

• So, after a C source file has been created, the programmer must **invoke the C compiler** before the program can be **executed** (**run**).

3 Stages of Compilation

Stage 1: **Preprocessing**

- Main purposes:
  - Centralize reused chunks of code
  - Allow “extensions” to the language
  - Make code more portable
- Performed by a program called the **preprocessor**
- Modifies the source code (in RAM) according to **preprocessor directives (preprocessor commands**) embedded in the source code
- The source code as stored on disk is not modified.
- “Include files” have names of form “*.h”

3 Stages of Compilation (con't)

Stage 2: **Compilation**

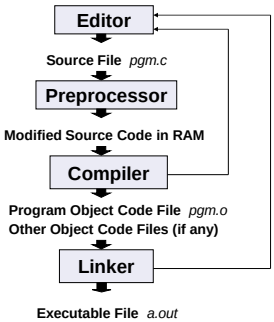
- Performed by a program called the **compiler**
- Translates the preprocessor-modified source code into **object code (machine code)**
- Checks for **syntax errors** and **warnings**
- Saves the object code to a disk file, if instructed to do so (we will not do this).
  - If any compiler errors are received, no object code file will be generated.
  - An object code file will be generated if only warnings, not errors, are received.

3 Stages of Compilation (con't)

Stage 3: **Linking**

- Combines the program object code with other object code to produce the executable file.
- The other object code can come from the **Run-Time Library**, other libraries, or object files that you have created.
- Saves the executable code to a disk file. On the Linux system, that file is called **a.out**.
  - If any linker errors are received, no executable file will be generated.

Program Development Using gcc



A Simple C Program

```
1.  /* Filename:      hello.c
2.   * Author:        Brian Kernighan & Dennis Ritchie
3.   * Date written:  ?/?/1978
4.   * Description:   This program prints the greeting
                          "Hello, World!"
5.   */
6.  #include  <stdio.h>
7.  int main ( )
8.  {
9.      printf ("Hello, World!\n") ;
10.     return 0 ;
11. }
```


Anatomy of a C Program

program header comment

preprocessor directives (if any)

```
int main ( )
{
    statement(s)
    return 0 ;
}
```


Program Header Comment

- A **comment** is descriptive text used to help a *reader* of the program understand its content.
- All comments must begin with the characters /* and end with the characters */
- These are called **comment delimiters**
- The program header comment always comes first.
- Look at the class web page for the required contents of our header comment.

Preprocessor Directives

- Lines that begin with a # in column 1 are called **preprocessor directives** (**commands**).
- Example: the **#include <stdio.h>** directive causes the preprocessor to include a copy of the standard input/output header file **stdio.h** at this point in the code.
- This header file was included because it contains information about the printf ( ) function that is used in this program.

int main ( )

- Every program must have a **function** called **main**. This is where program execution begins.
- main() is placed in the source code file as the first function for readability.
- The **reserved word** “int” indicates that main() **returns** an integer value.
- The parentheses following “main” indicate that it is a function.

The Function Body

- A left brace (curly bracket) -- **{** -- begins the **body** of every function. A corresponding right brace -- **}** -- ends the function body.
- The style is to place these braces on separate lines in column 1 and to indent the entire function body 3 to 4 spaces.

printf (“Hello, World!\n”) ;

- This line is a C **statement**.
- It is a **call** to the function **printf ( )** with a single **argument (parameter)**, namely the **string** “Hello, World!\n”.
- Even though a string may contain many characters, the string itself should be thought of as a single quantity.
- Notice that this line ends with a semicolon. All statements in C end with a semicolon.

return 0 ;

- Because function main() returns an integer value, there must be a statement that indicates what this value is.
- The statement

  return 0 ;

  indicates that main() returns a value of zero to the operating system.
- A value of 0 indicates that the program successfully terminated execution.
- Do not worry about this concept now. Just remember to use the statement.

Another C Program

```
1. /*************************************************************
2. ** File: message.c
3. ** Author: Joe Student
4. ** Date: 9/15/06
5. ** Section: 0101
6. ** E-mail: jstudent22@umbc.edu
7. **
8. ** This program prints a cool message to the user.
9. *************************************************************/
```

Another C Program (con't)

```
10. #include <stdio.h>
11. int main()
12. {
13.     printf("Programming in CMSC104 is\nfun. ") ;
14.     printf("C is a really cool language!\n") ;
15.     return 0 ;
16. }
```

What will the output be?

Using the C Compiler at UMBC

- Invoking the compiler is system dependent.
  - At UMBC, we have two C compilers available, **cc** and **gcc**.
  - For this class, we will use the gcc compiler as it is the compiler available on the Linux system.

Invoking the gcc Compiler

At the prompt, type

```
gcc -Wall program.c -o program.out
```

where ***program.c*** is the C program source file.

- **-Wall** is an option to turn on all compiler **warnings** (best for new programmers).

The Result : a.out

- If there are no errors in pgm.c, this command produces an **executable file**, which is one that can be executed (run).
- If you do not use the "-o" option, the compiler names the executable file **a.out** .
- To execute the program, at the prompt, type

  program.out
- Although we call this process "compiling a program," what actually happens is more complicated.

Good Programming Practices

- C programming standards and indentation styles are available on the 104 course Web page.
- You are expected to conform to these standards for all programming projects in this class and in CMSC 201. (This will be part of your grade for each project!)
- The program just shown conforms to these standards, but is uncommented (we'll discuss commenting your code later).
- Subsequent lectures will include more "Good Programming Practices" slides.